Süsteemprogrammeerimine keeles C

Kompileerimine ja utiliidindus

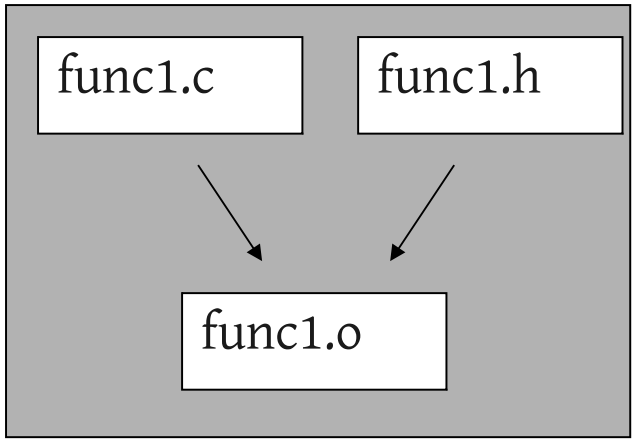
C Preprotsessor

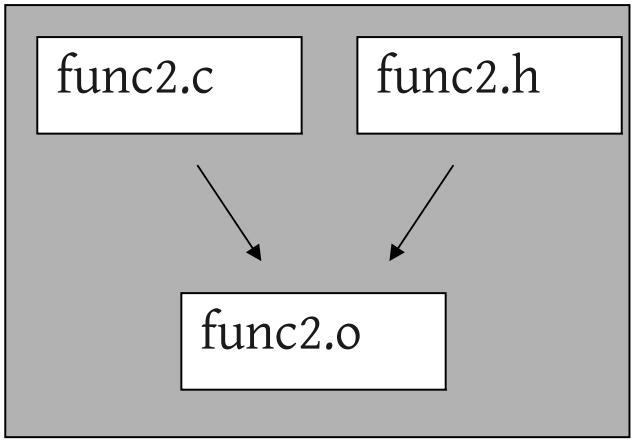
- Programm lastakse enne kompileerimist preprotsessorist läbi
- Preprotsessori direktiivid:
  - #include – lisab *header* faili (või suvalise faili)
  - #define – defineerib makro või konstandi
  - #ifndef, #ifdef, #endif – tingimuslik lisamine
  - #if, #else, #elif, #endif
  - #undef – tühistab definitsiooni

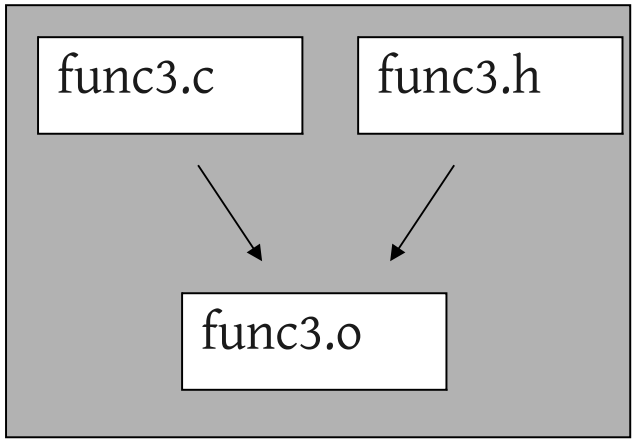
Header failid

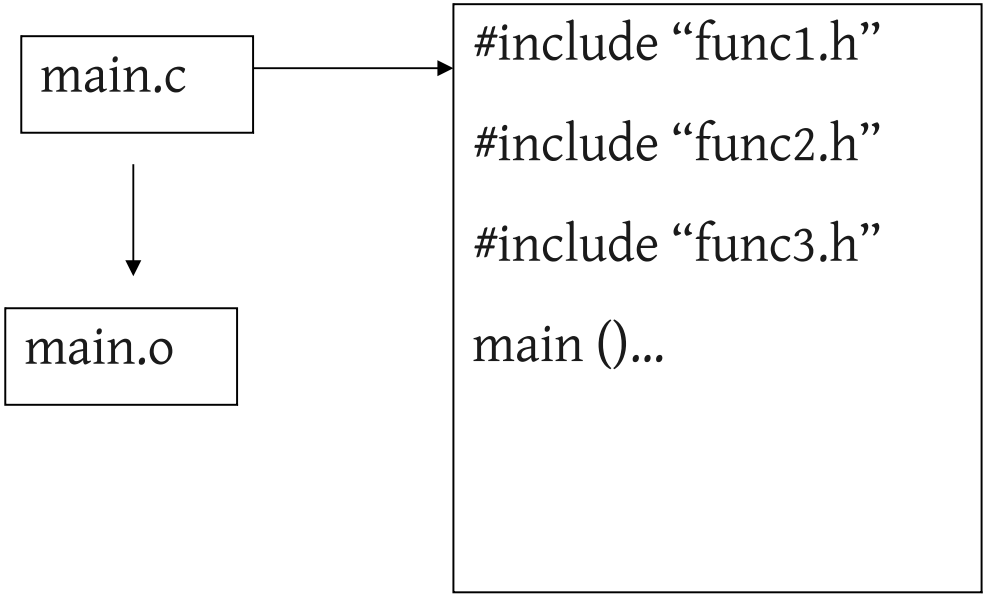
- Milleks on header failid?
- Lihtne viis kasutada varem defineeritud funktsioone (sisaldavad ainult prototüüpi)
- Modulaarsus: võimaldab kirjutada väikseid komponente ja neid ühendada
- Iga komponent omab **.c** ja **.h** faili
  - .c failis on funktsioonide definitsioonid
  - .h failis on funktsioonide prototüübid, konstandid, makrode definitsioonid
- Lihtsam siluda ja kasutada

Programmi struktuur

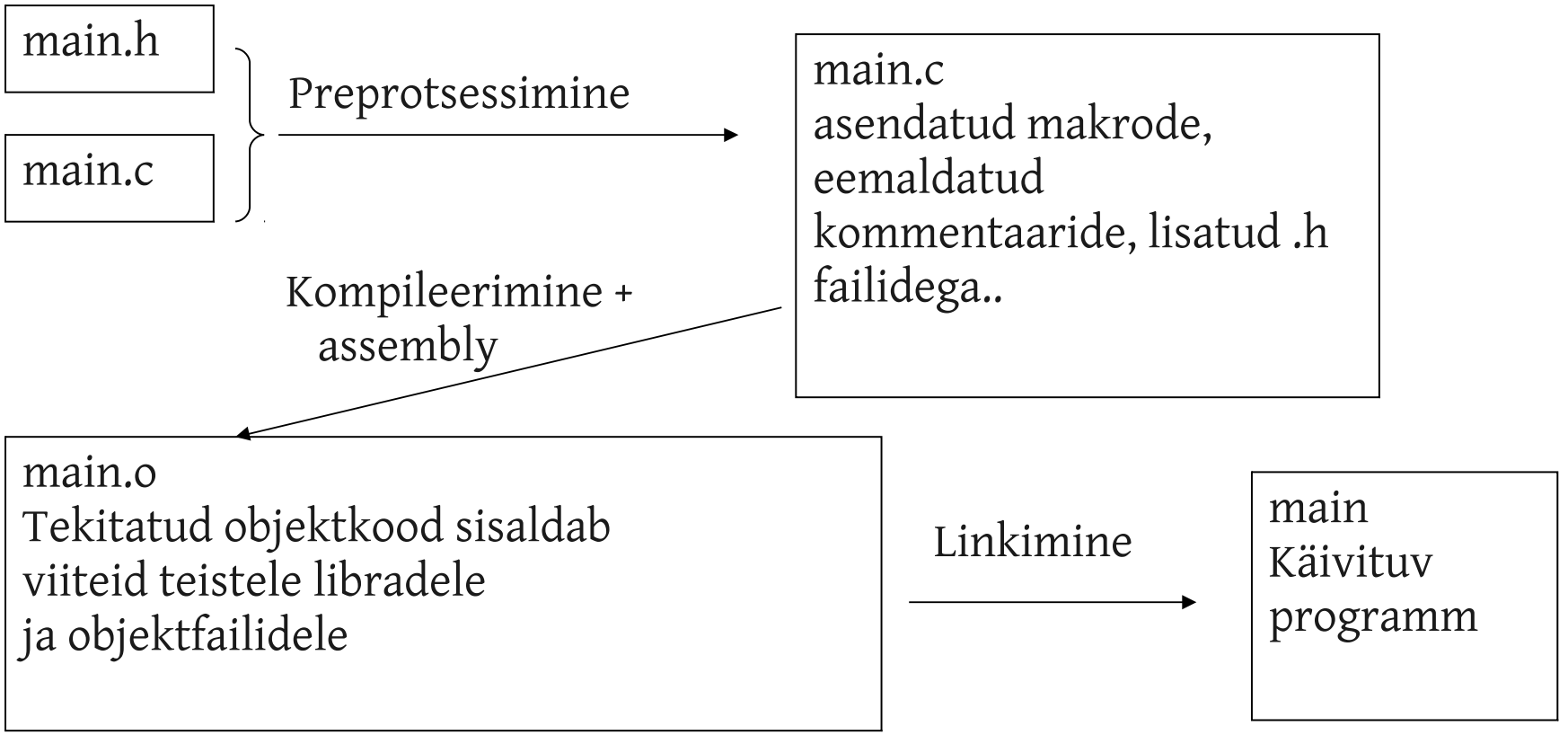








Programmi ehitamine



Tingimuslik teksti lisamine

- *Header* failid kasutavad tihti #ifndef käsku
- *Header*it mitu korda lisada on enamasti väga halb (pl.c lisab h2.h ja h1.h , mis samuti lisab h2.h)
- Selle vältimiseks tehakse *headeris* järgmine trikk:

```
#ifndef _header_name
#define _header_name

/* teiskordsel saabumisel seda siin ei täideta
  */
…

#endif
```

Mitmefaililise proge näide

main.c

```
#include "func1.h"
#include <stdio.h>
main()
{
   printf ("square of %d is
   %d\n", 2, sqr(2));
}
```

func1.c

```
#include "func1.h"
int sqr(int x)
{
    return x * x;
}
```

func1.h

```
#ifndef _FUNC1_H_
#define _FUNC1_H_
int sqr(int x);
#endif // _FUNC1_H_
```

extern mäluklass

- Kui on vaja jagada mõnd globaalmuutujat mitme faili (mooduli) vahel ei piisa vaid sama nimega nimetamisest.
  extern int errno;
- Extern võtmesõna teatab, et muutuja aadress tuleb otsida kompileerimise ajal mõnest teisest failist.

errno

- Veanumber, mille enamik standardteegi funktsioonidest tagastab.
- Annab vea kohta rohkem infot
  Tõlgendamiseks funktsioon: strerror(), perror() või error()
- Veanumbreid võite ka ise muuta ja tagastada

gcc peatamise näited

- gcc -E – jooksuta preprotsessor
- gcc -c – objektfailid (.o)
- gcc -S – assemblerfailid (.s)

Veel gcc võtmeid

- -L*kataloginimi* – teegi asukoha määramine
- -lm – matemaatikateegi kaasamine
- -O1, -O2. -O3 – optimeeri (vastavalt agressiivsemalt)
- -g – säilita sümboltabel (debugimiseks)

Make

- Kirjelda oma programm
  eesmärk: eeldusfailid
  [tab] käsud eeldusest eesmärgi loomiseks

```
all: test

test:test.o
    gcc test.o -o test

test.o:test.c
    gcc test.c -c -o test.o
```

Hexdump

- Vahend binaarsete failide vaatlemiseks

gdb

- Gnu DeBugger
- kompileerimisel -g võti
- run jooksutab käsu
- näitab millisel real kokku jooksis ilma printf õuduseta
- võimaldab (suvalist!) töötavat programmi uurida
- pöördkompileerida
- Olemas (üsna kole) graafiline liides - ddd

tar

- Lähtetekste jagatakse tihti tar.gz failides
- Lahti-ja kokkupakkimine ja tihendamine
- tar -czf failinimi.tar.gz kataloog
- tar -xzf failinimi.tar.gz

shell

- Kest on see programm millesse sisselogides satud
  sh
  tcsh
  bash
- Käivitab teisi programme ja tõlgendab käsurida

Programmi käsureavalikud

- Kokkuleppeliselt tähistatakse programmi valikud käsureal miinusmärgiga (-) ja ühe tähega (ei ole päris tõsi, aga arvake praegu et on)
- Valikuid võib üksteise otsa kirjutada (kui neil pole parameetreid)
- Valikutel võivad olla parameetreid

```
myprog -a -b
myprog -ab
myprog -ba

myprog2 -i infile -ooutfile
```

- Kuidas sellises situatsioonis laisk olla?

getopt()

```
#include <unistd.h>
int opterr;
int optopt;
int optind;
char * optarg;
int getopt(int argc, char**argv, const char *options)
```

- Tagastab valikute nimekirjast järgmise
- options näitab ära võimalikud "lubatud" käsureavalikud, täht võimaliku tähemärgi, sellele järgnev koolon (:), et järgneb argument, ja (::) kui argument pole kohustuslik
- opterr: kui määrata 0, ei näita vigu välja
- optind: järgmine valik

getopt() (2)

- getopt() vahetab tavaolukorras argv sees olevate valikute järjekorda kuni kõik miinuseta on lõpus
- Kui rohkem valikuid pole, tagastub -1
- Valikute lõppemisel saab kontrollida üle jäänud argumente võrreldes argc-d ja optind muutujat.
- Tundmatu valiku puhul tagastub '?', tundmatu märk salvestatakse muutujasse optopt

```
getopt(argc, argv, "abc:");
```

getopt() (3)

- Reeglina kutsutakse välja tsüklis, mis lõpetab, kui getopt() tagastab -1
- Tagastatavat väärtust valitakse enamasti switch-lausega, kus väärtustatakse iga valiku puhul muutuja, mis hiljem programmi tööd vastavalt mõjutab
- Ülejäänud programmiargumentide jaoks kasutatakse teist tsüklit

Näide

```
#include <unistd.h>
#include <stdio.h>

int
main (int argc, char **argv)
{
  int aflag = 0;
  int bflag = 0;
  char *cvalue = NULL;
  int index;
  int c;

  opterr = 0;

  while ((c = getopt (argc, argv, "abc::")) != -1)
```

Näide (2)

```
 switch (c) {
  case 'a':
  aflag = 1;
  break;
 case 'b':
  bflag = 1;
  break;
 case 'c':
  cvalue = optarg;
  break;
 case '?':
    if (isprint (optopt))
     fprintf (stderr, "Unknown option `-%c'.\n", optopt);
    else
     fprintf (stderr, "Unknown option char `\\x%x'.\n",
                  optopt);
    return 1;
 default:
  abort();
}
```

Näide (3)

```
    printf ("aflag = %d, bflag = %d, cvalue = %s\n",
            aflag, bflag, cvalue);

    for (index = optind; index < argc; index++)
      printf ("Non-option argument %s\n", argv[index]);
    return 0;
  }
```

GNU täiendused

- GNU projekti raames on libc-s ka getopt_long
- Selle eesmärgiks on programmide õppimine lihtsamaks teha:

```
ls -Ah
ls --almost-all --human-readable
```

- Pikad valikud algavad topeltmiinusega (--)
- Pika valiku argumendid on kujul:

  --valikunimi=väärtus

- unistd.h asemel on prototüüp eraldi getopt.h failis

struct option

- getopt_long() võtab argumendiks massiivi valikuid kirjeldavatest structidest

```
struct option  {
   const char *name;
   int has_arg;
   int *flag;
   int val;
}
```

- name: valiku nimi
- has_args: no_argument, required_argument või optional_argument
- flag ja val osas juurdleme põhjalikumalt kohe

struct option (2)

- flag:
  - kui on pointer mingile väärtusele, salvestatakse **`var`** sisu näidatud aadressile, kui valik peaks ette juhtuma
  - kui on 0, on **`var`** mingi unikaalne number, mis valikut tuvastab (enamasti sama, mis oleks tavalise getopti puhul; pärast struct optioni näidet selgub, miks see hästi loogiline on)
- Massiivi viimane element peab olema struct option, mille kõik elemendid on 0

struct option-i näide

```
static struct option long_options[] = {
   /* siin on flag määratud. */
     {"lamiseja", no_argument, &verbose_flag, 1},
     {"tasane",   no_argument, &verbose_flag, 0},
   /* siin on flag 0 .
      eristamine käib tähtede/järjekorranr-i abil. */
     {"add",     no_argument,       0, 'a'},
     {"append",  no_argument,       0, 'b'},
     {"delete",  required_argument, 0, 'd'},
     {"create",  required_argument, 0, 'c'},
     {"file",    required_argument, 0, 'f'},
     {0, 0, 0, 0}
};
```

getopt_long()

- Töötab lühikestega täpselt nagu tavaline getopt()
- Pikkadega sõltub valiku flag parameetrist: kas salvestab kuskile muutuja või tagastab väärtuse val
  - seega kui tagastame lühikese valiku tähe, saame kasutada ka pika puhul sama käitumist

```
int getopt_long (int argc, char *const *argv,
                 const char *shortopts,
                 const struct option *longopts,
                 int *indexptr)
```

- Kui ülaltoodust ei piisa, on olemas argp(), mis meenutab karujahti keskmaaraketiga: väga tõhus, aga sihtimine on veidi keeruline

getopt_long() näidis

```
c = getopt_long (argc, argv, "abc:d:f:",
                 long_options, &option_index);
    if (c == -1) /* valikute ots */
        break;
    switch (c) {
      case 0:  /* kui lipp, ignoreerime praegu. */
        if (long_options[option_index].flag != 0)
                break;
        printf ("option %s",
                  long_options[option_index].name);
        if (optarg)
            printf (" with arg %s", optarg);
        printf ("\n");
        break;
      case 'a':
        puts ("option -a\n");
        break;
      case 'b':
        /* ... */
```

Getopt lõpetuseks

- getopt() on üsna lihtne, mugav ja veakindel
- getopt_long() on tunduvalt keerulisem, kuid on ikkagi mugavam, kui ise analoogne parsija kirjutada
- Kasutage!